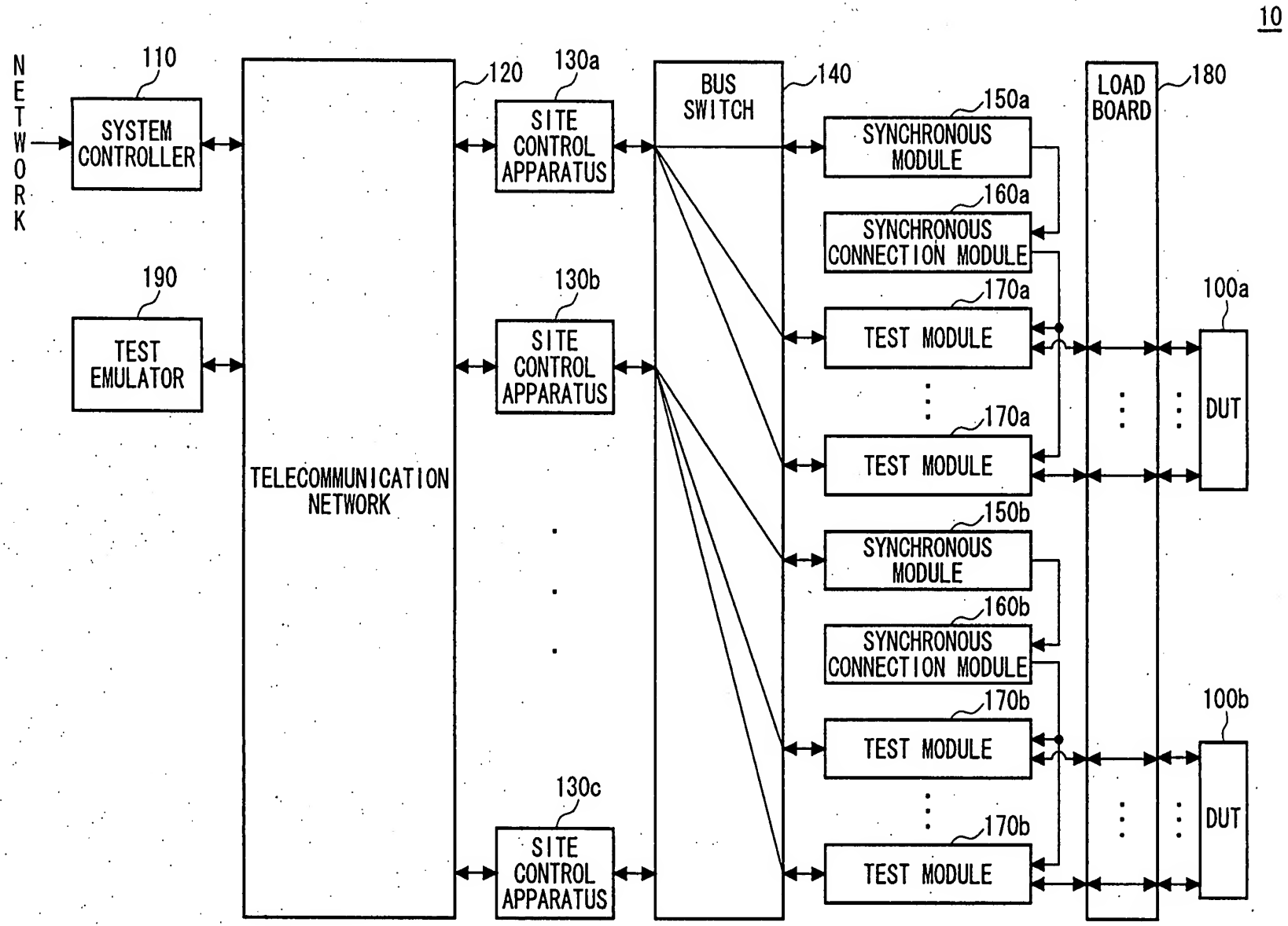

10
NETWORK
110
SYSTEM CONTROLLER
190
TEST EMULATOR
TELECOMMUNICATION NETWORK
120
130a
SITE CONTROL APPARATUS
130b
SITE CONTROL APPARATUS
130c
SITE CONTROL APPARATUS
BUS SWITCH
140
150a
SYNCHRONOUS MODULE
160a
SYNCHRONOUS CONNECTION MODULE
170a
TEST MODULE
170a
TEST MODULE
150b
SYNCHRONOUS MODULE
160b
SYNCHRONOUS CONNECTION MODULE
170b
TEST MODULE
170b
TEST MODULE
LOAD BOARD
180
100a
DUT
100b
DUT

FIG. 1

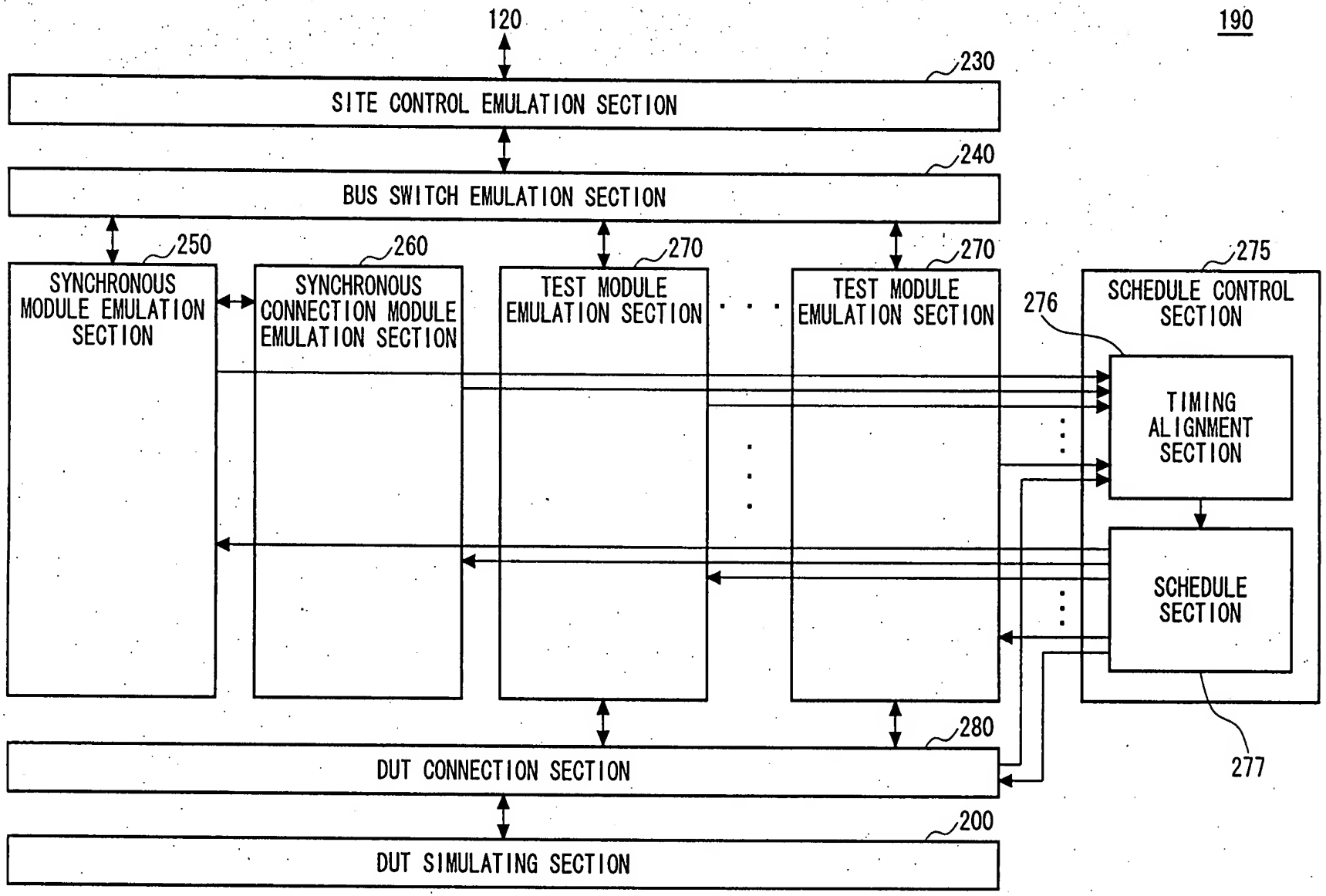

120
190
230
SITE CONTROL EMULATION SECTION
240
BUS SWITCH EMULATION SECTION
250
SYNCHRONOUS MODULE EMULATION SECTION
260
SYNCHRONOUS CONNECTION MODULE EMULATION SECTION
270
TEST MODULE EMULATION SECTION
270
TEST MODULE EMULATION SECTION
275
SCHEDULE CONTROL SECTION
276
TIMING ALIGNMENT SECTION
SCHEDULE SECTION
277
280
DUT CONNECTION SECTION
200
DUT SIMULATING SECTION

FIG. 2

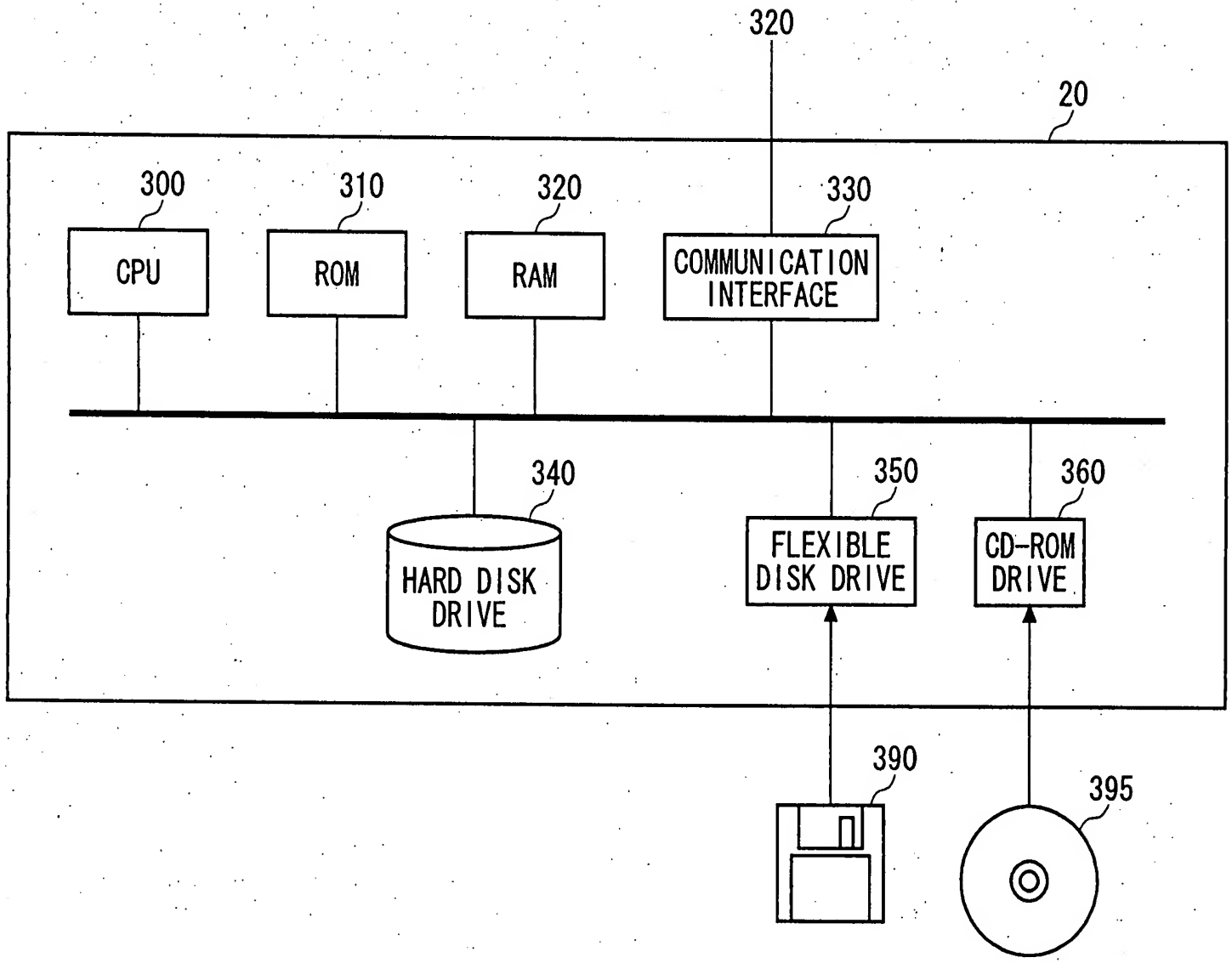

320
20
300
310
320
330
CPU
ROM
RAM
COMMUNICATION INTERFACE
340
350
360
HARD DISK DRIVE
FLEXIBLE DISK DRIVE
CD-ROM DRIVE
390
395

FIG. 3

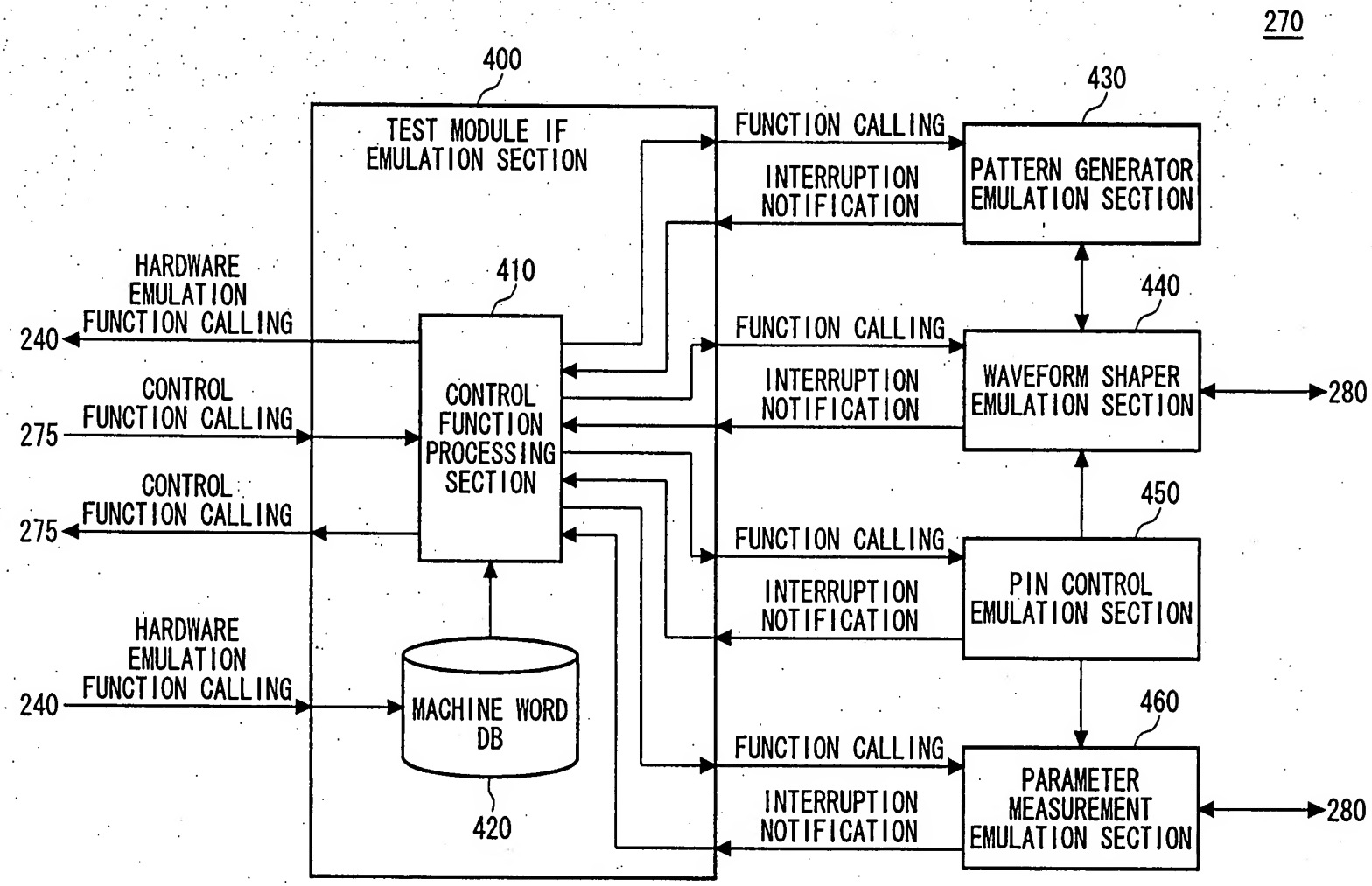

270
400
TEST MODULE IF EMULATION SECTION
410
CONTROL FUNCTION PROCESSING SECTION
MACHINE WORD DB
420
HARDWARE EMULATION FUNCTION CALLING
240
CONTROL FUNCTION CALLING
275
CONTROL FUNCTION CALLING
275
HARDWARE EMULATION FUNCTION CALLING
240
FUNCTION CALLING
INTERRUPTION NOTIFICATION
430
PATTERN GENERATOR EMULATION SECTION
440
WAVEFORM SHAPER EMULATION SECTION
280
450
PIN CONTROL EMULATION SECTION
460
PARAMETER MEASUREMENT EMULATION SECTION
280

FIG. 4

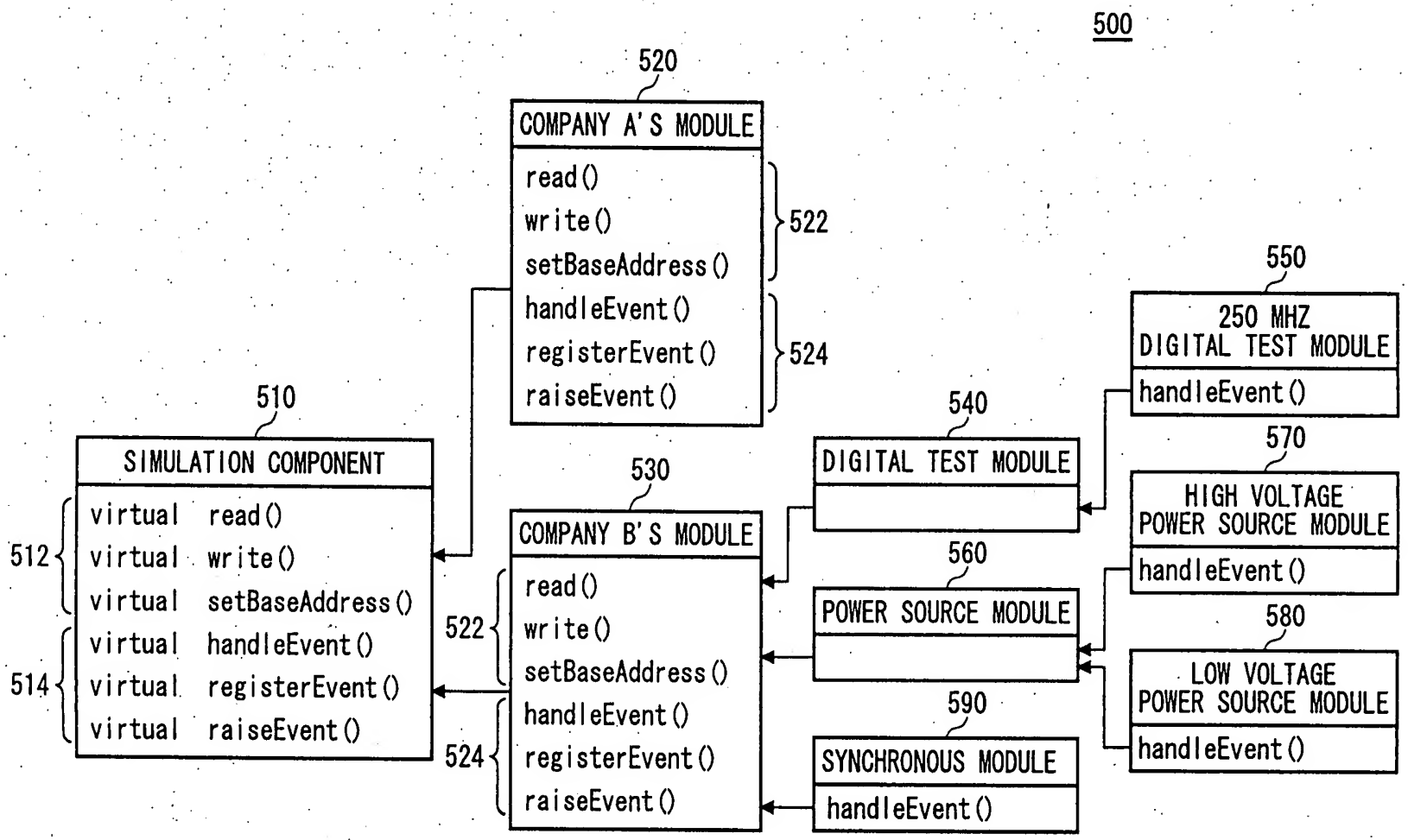

500
520
COMPANY A'S MODULE
read()
write()
setBaseAddress()
handleEvent()
registerEvent()
raiseEvent()
522
524
550
250 MHZ
DIGITAL TEST MODULE
handleEvent()
510
SIMULATION COMPONENT
virtual read()
virtual write()
virtual setBaseAddress()
virtual handleEvent()
virtual registerEvent()
virtual raiseEvent()
512
514
530
COMPANY B'S MODULE
read()
write()
setBaseAddress()
handleEvent()
registerEvent()
raiseEvent()
522
524
540
DIGITAL TEST MODULE
570
HIGH VOLTAGE
POWER SOURCE MODULE
handleEvent()
560
POWER SOURCE MODULE
580
LOW VOLTAGE
POWER SOURCE MODULE
handleEvent()
590
SYNCHRONOUS MODULE
handleEvent()

FIG. 5

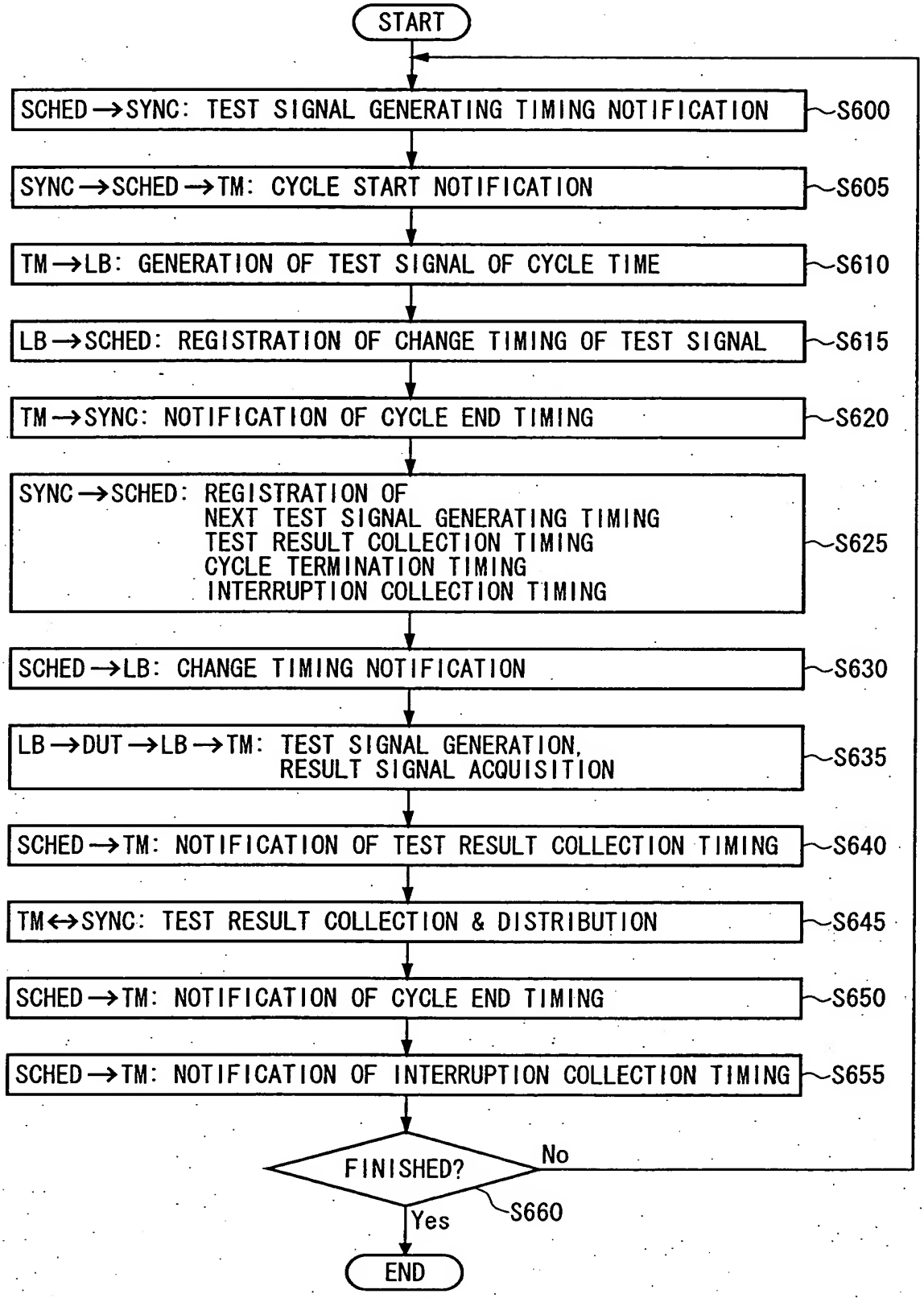

START
SCHED→SYNC: TEST SIGNAL GENERATING TIMING NOTIFICATION
S600
SYNC→SCHED→TM: CYCLE START NOTIFICATION
S605
TM→LB: GENERATION OF TEST SIGNAL OF CYCLE TIME
S610
LB→SCHED: REGISTRATION OF CHANGE TIMING OF TEST SIGNAL
S615
TM→SYNC: NOTIFICATION OF CYCLE END TIMING
S620
SYNC→SCHED: REGISTRATION OF
NEXT TEST SIGNAL GENERATING TIMING
TEST RESULT COLLECTION TIMING
CYCLE TERMINATION TIMING
INTERRUPTION COLLECTION TIMING
S625
SCHED→LB: CHANGE TIMING NOTIFICATION
S630
LB→DUT→LB→TM: TEST SIGNAL GENERATION,
RESULT SIGNAL ACQUISITION
S635
SCHED→TM: NOTIFICATION OF TEST RESULT COLLECTION TIMING
S640
TM↔SYNC: TEST RESULT COLLECTION & DISTRIBUTION
S645
SCHED→TM: NOTIFICATION OF CYCLE END TIMING
S650
SCHED→TM: NOTIFICATION OF INTERRUPTION COLLECTION TIMING
S655
FINISHED?
No
S660
Yes
END

FIG. 6

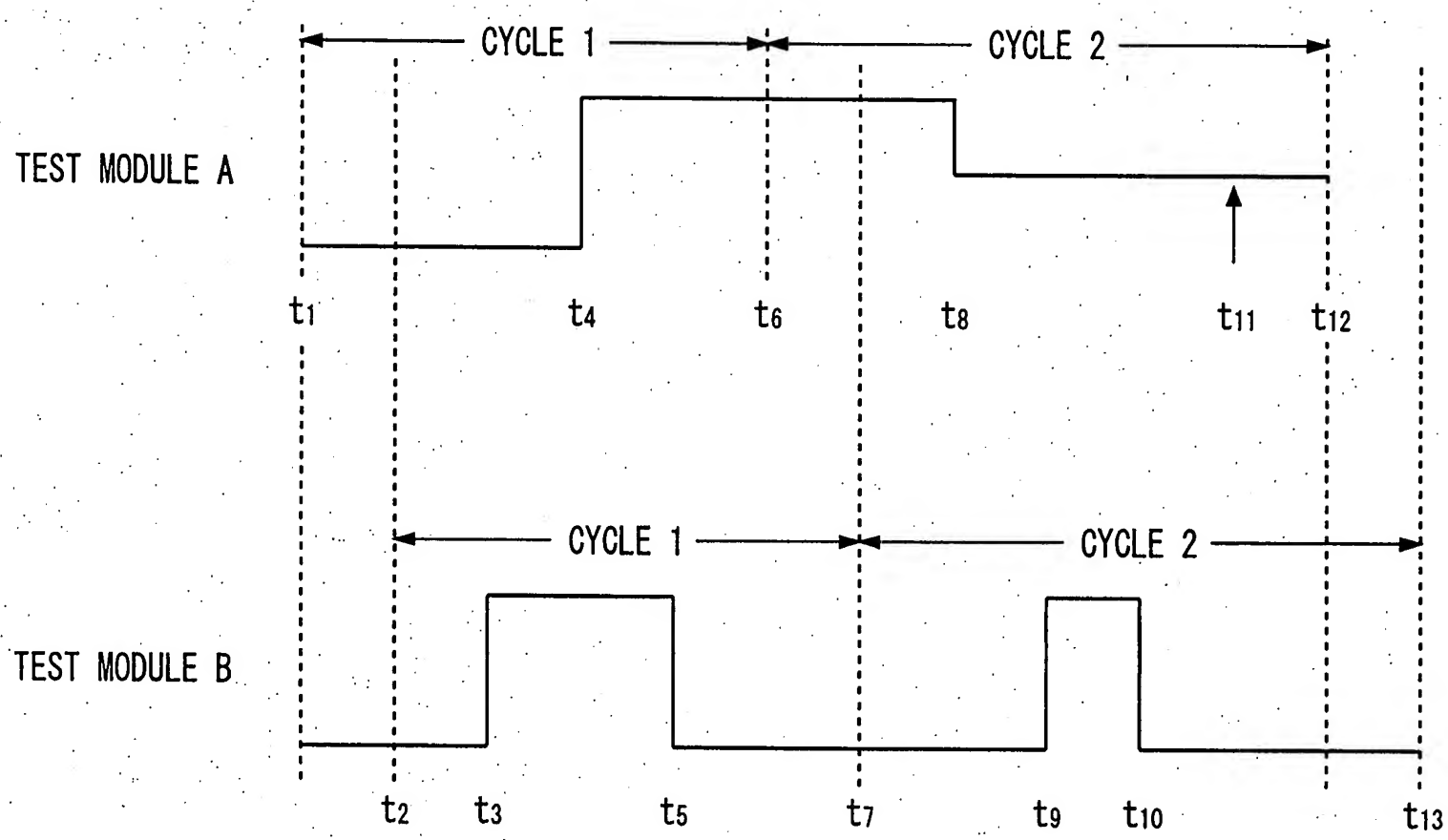

CYCLE 1
CYCLE 2
TEST MODULE A
t1
t4
t6
t8
t11
t12
CYCLE 1
CYCLE 2
TEST MODULE B
t2
t3
t5
t7
t9
t10
t13

FIG. 7

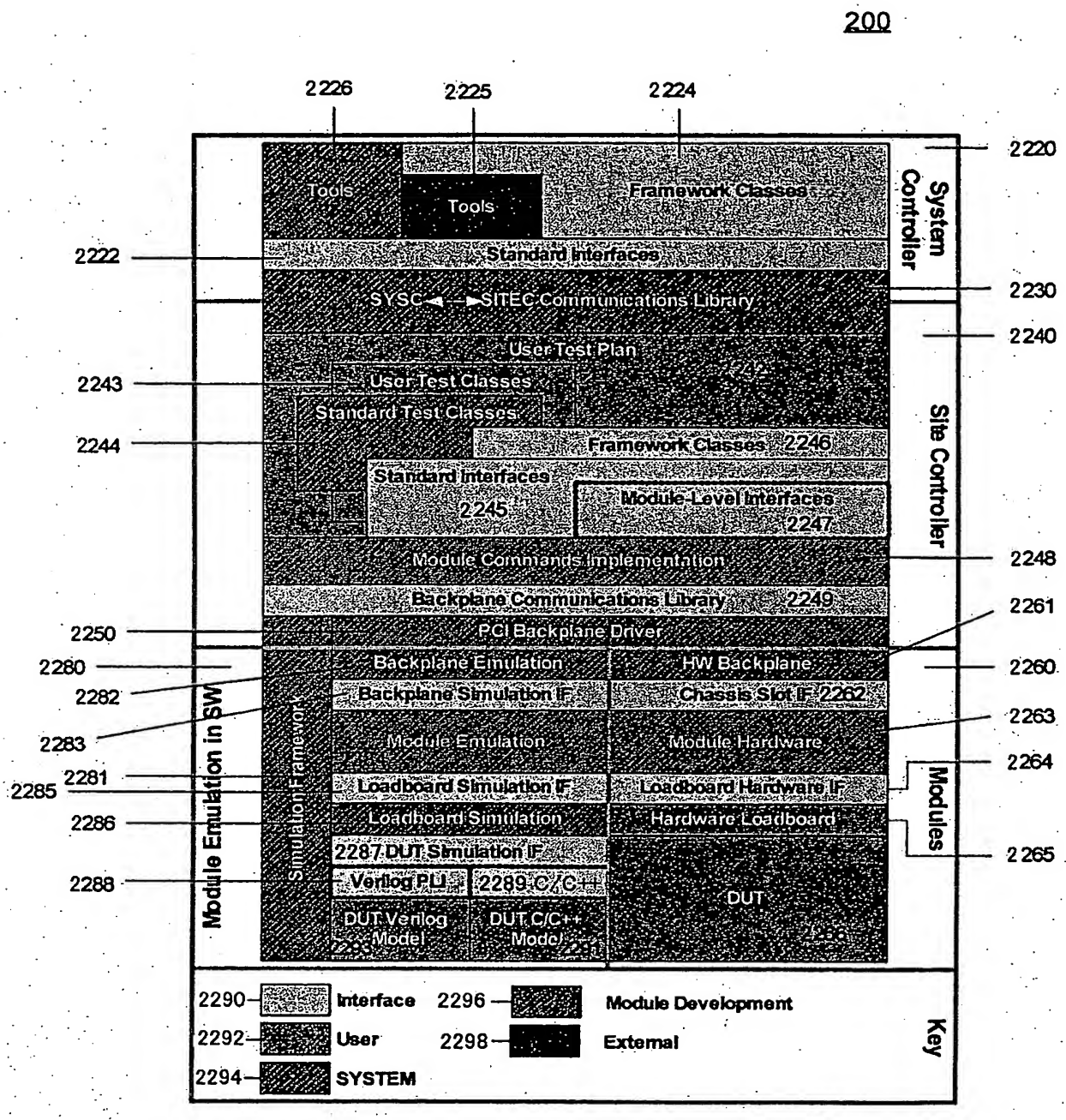


Figure 8

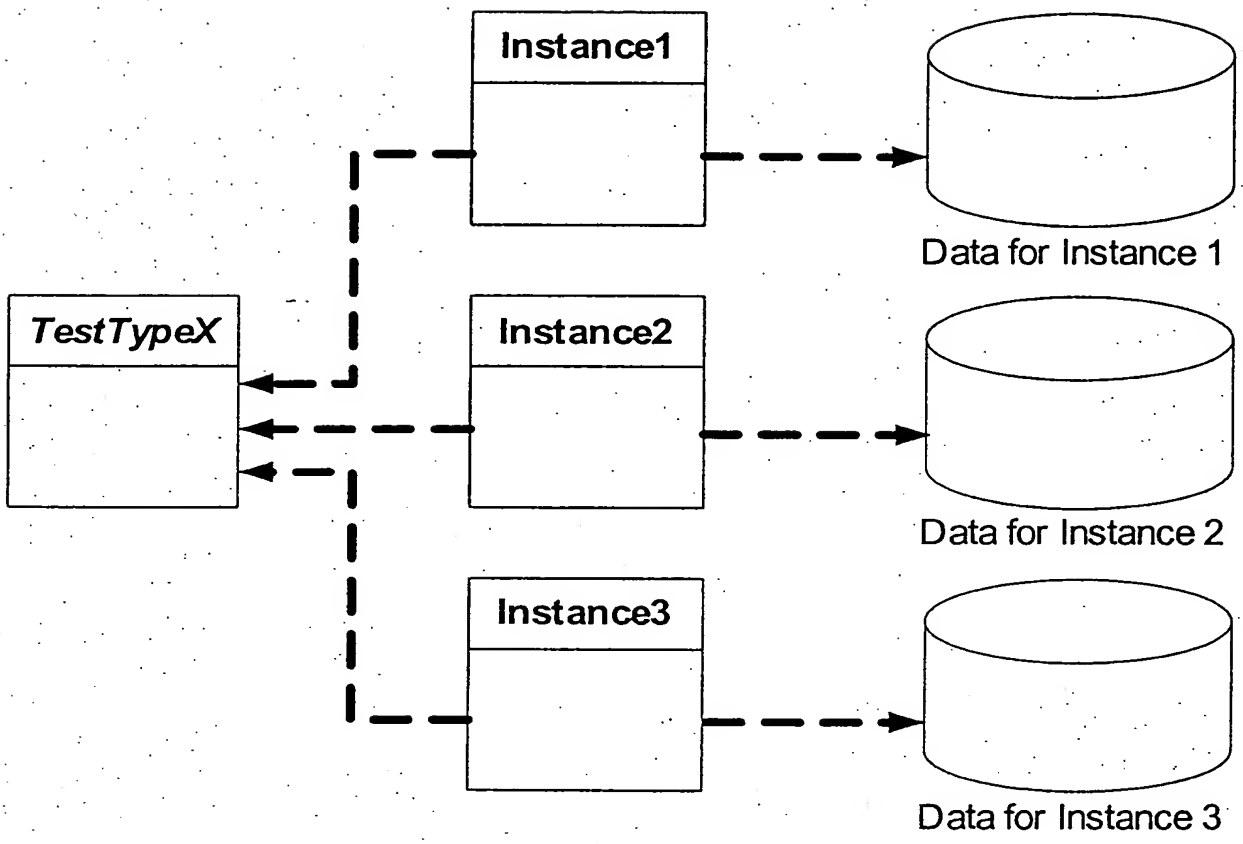

Instance1
Data for Instance 1
TestTypeX
Instance2
Data for Instance 2
Instance3
Data for Instance 3

Figure 4~~9~~

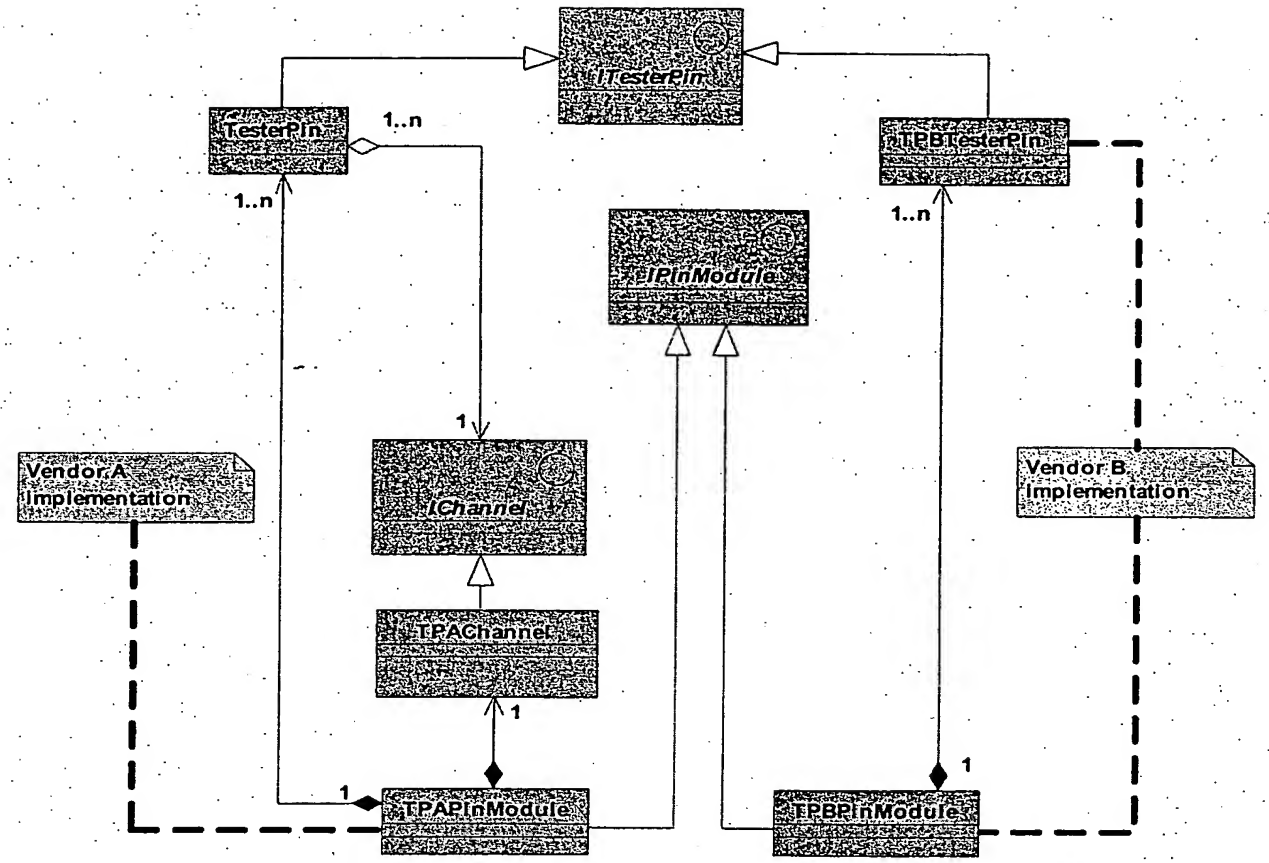


Figure 5/10

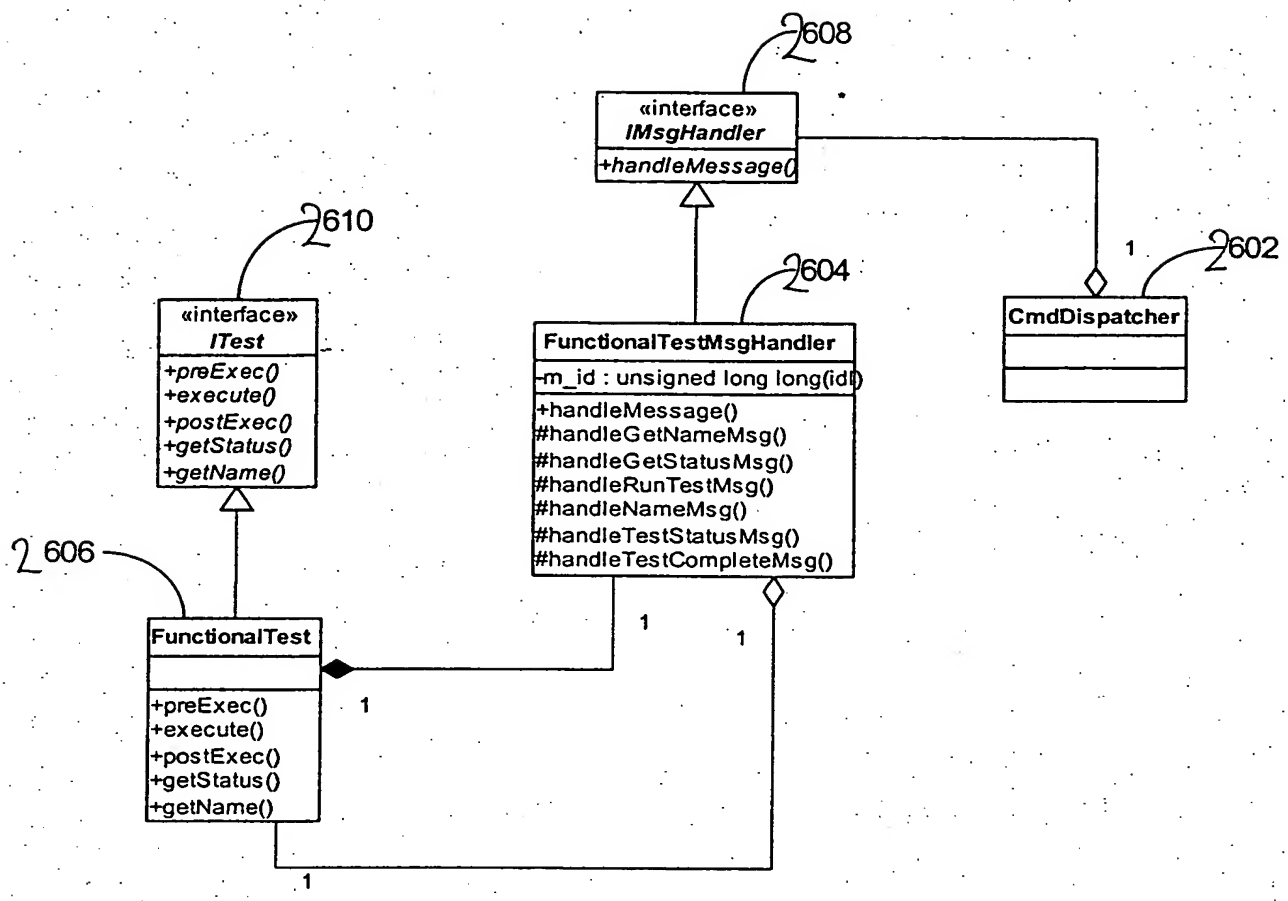

2608
«interface»
IMsgHandler
+handleMessage()
2610
«interface»
ITest
+preExec()
+execute()
+postExec()
+getStatus()
+getName()
2604
FunctionalTestMsgHandler
-m_id : unsigned long long(id)
+handleMessage()
#handleGetNameMsg()
#handleGetStatusMsg()
#handleRunTestMsg()
#handleNameMsg()
#handleTestStatusMsg()
#handleTestCompleteMsg()
2602
CmdDispatcher
2606
FunctionalTest
+preExec()
+execute()
+postExec()
+getStatus()
+getName()
1
1
1
1
1

Figure 6

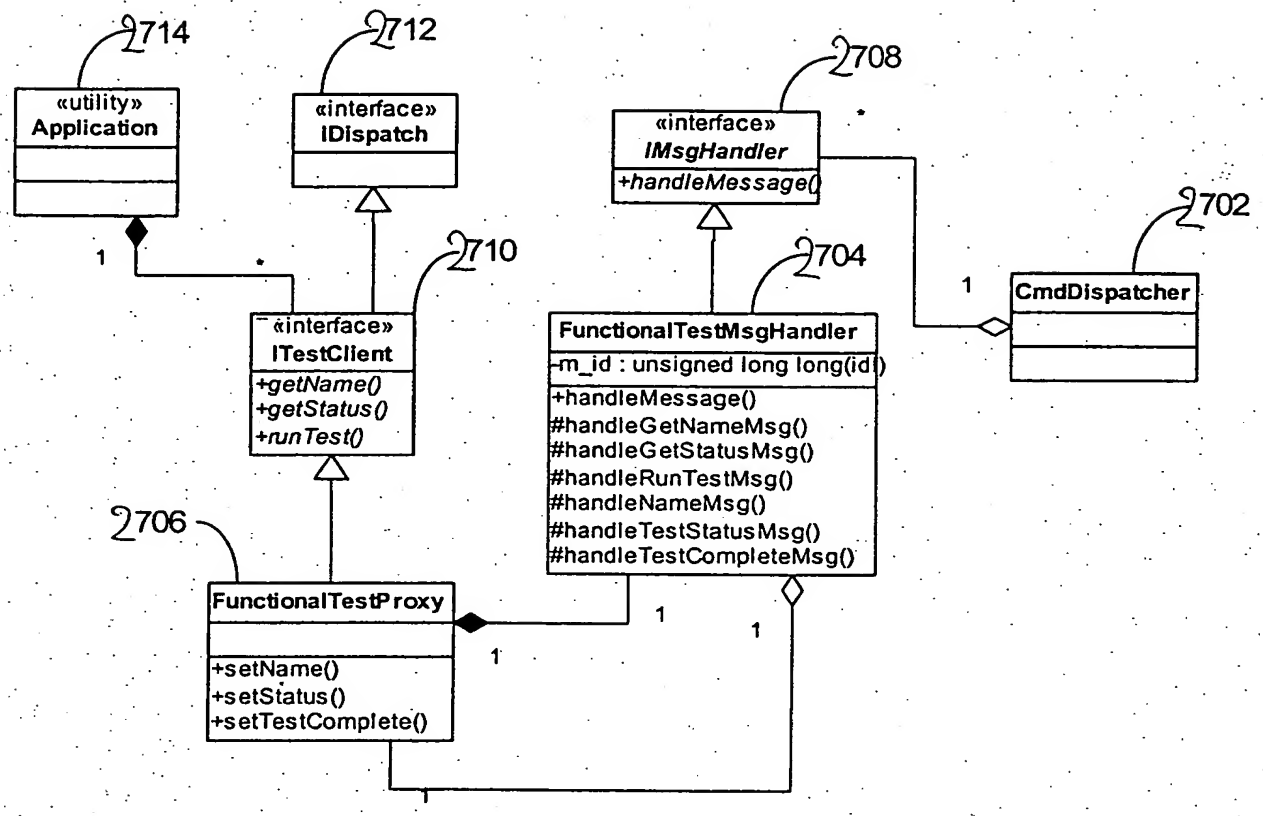

714
712
708
«utility»
Application
«interface»
IDispatch
«interface»
IMsgHandler
+handleMessage()
702
1
*
710
704
«interface»
ITestClient
+getName()
+getStatus()
+runTest()
FunctionalTestMsgHandler
-m_id : unsigned long long(id)
+handleMessage()
#handleGetNameMsg()
#handleGetStatusMsg()
#handleRunTestMsg()
#handleNameMsg()
#handleTestStatusMsg()
#handleTestCompleteMsg()
CmdDispatcher
706
FunctionalTestProxy
+setName()
+setStatus()
+setTestComplete()

Figure 7/12

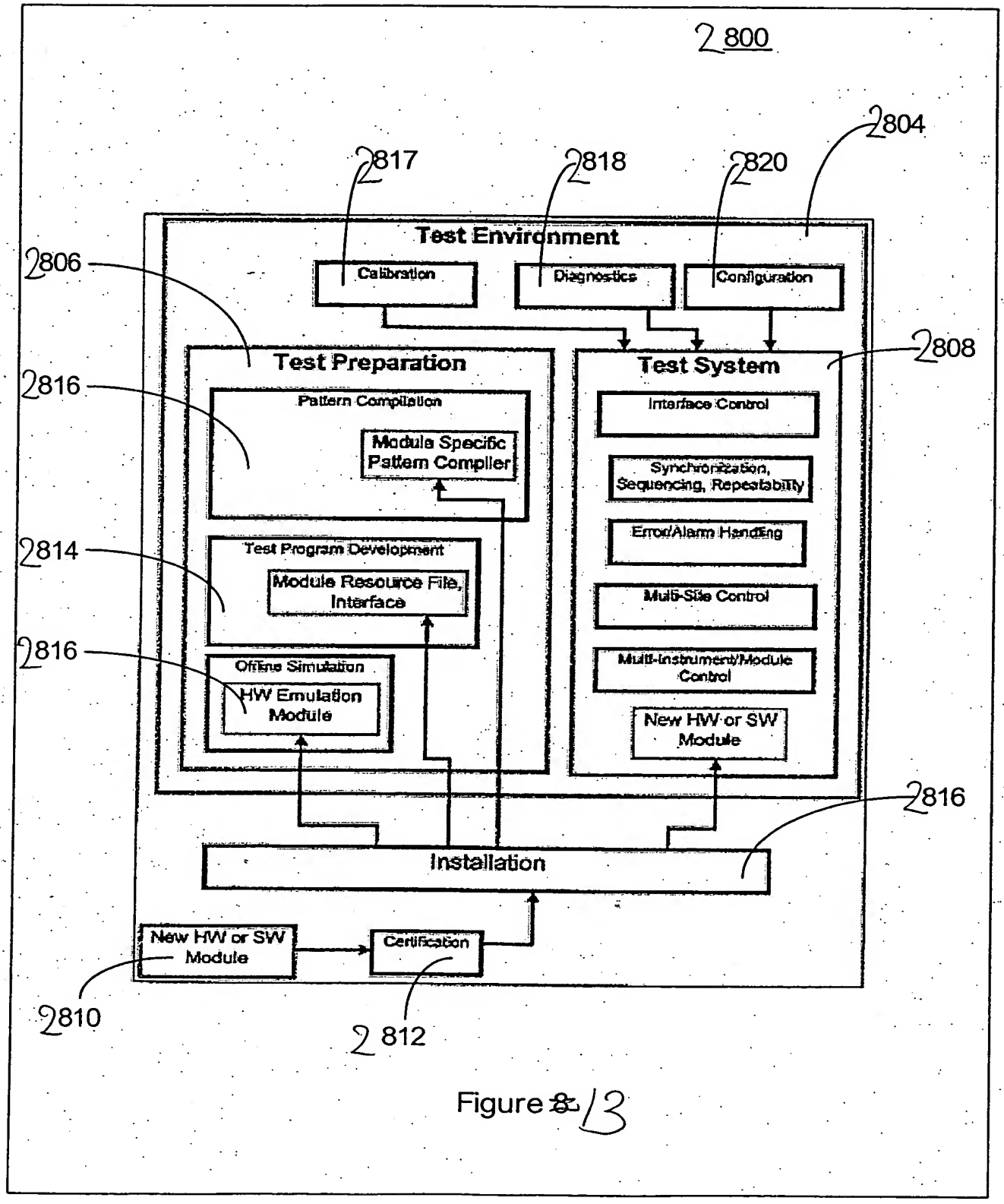

2800
2804
2817
2818
2820
Test Environment
Calibration
Diagnostics
Configuration
2806
Test Preparation
Test System
2808
2816
Pattern Compilation
Module Specific Pattern Compiler
Interface Control
Synchronization, Sequencing, Repeatability
Error/Alarm Handling
2814
Test Program Development
Module Resource File, Interface
Multi-Site Control
2816
Offline Simulation
HW Emulation Module
Multi-Instrument/Module Control
New HW or SW Module
2816
Installation
New HW or SW Module
Certification
2810
2812

Figure 13

~~図14~~ Fig. 14

5000

```
Global
{
  InitVoltage 0.5 V;        # Required. Initial voltage on all
                            # wires.
  RecoveryRate 2.5 V/nS;    # Optional. For drive conflict in
                            # analog signals.
}
```

5010

```
# Module Emulator.
EMUModule "module1"         # Module DLL.
{

  Waveform                  # Optional. Resource declaration.
  {
    Step 1-32, 35;          # Step type waveforms on channels 1 thru 32
                            # channel 35.
    Analog 33, 34;          # Analog waveform on channels 33 and 34.
  }
  Port 1                    # Declares the GBUS Port for this module.
  {
    SerialNumber 1;         # Required. Should match setting in
                            # Module Configuration File.
    ProductRevision 1;      # Required. Should match setting in
                            # Module Configuration File.
    Params                  # To be passed to DLL.
    {
      test  "param1";
      key   "abc";
    }
  }
  Port 8
  {
    LogicalPort 3;          # Optional. Designate Logical Port to use
                            # in offline configuration file.
                            # Default is the GBUS port.
    SerialNumber 2;
    ProductRevision 1;
    Params                  # To be passed to DLL.
    {
      test  "param1";
      key   "abc";
    }
  }
}
```

5020

~~図15~~ Fig. 15

5000

```
# Module Emulator
EMUModule "module2"
{
  Waveform
  {
    Step 1-32;
  }
  Port 2
  {
    SerialNumber 1;
    ProductRevision 1;
  }
}
```

5020

```
# Module Emulator
EMUModule "dps"
{
  Waveform
  {
    Slew 1-32 @ 2.0 V / nS; # The slew rate is required
                            # for all slewing waveforms.
  }
  Port 4
  {
    SerialNumber 1;
    ProductRevision 1;
  }
}
```

5010

~~図16~~ Fig. 16

5100

```
Global
{
  RegSelect "PatTraceSel";  # Pattern Tracing - Name of OASIS
                            # Simulated Register Selection File.
}
```

5110

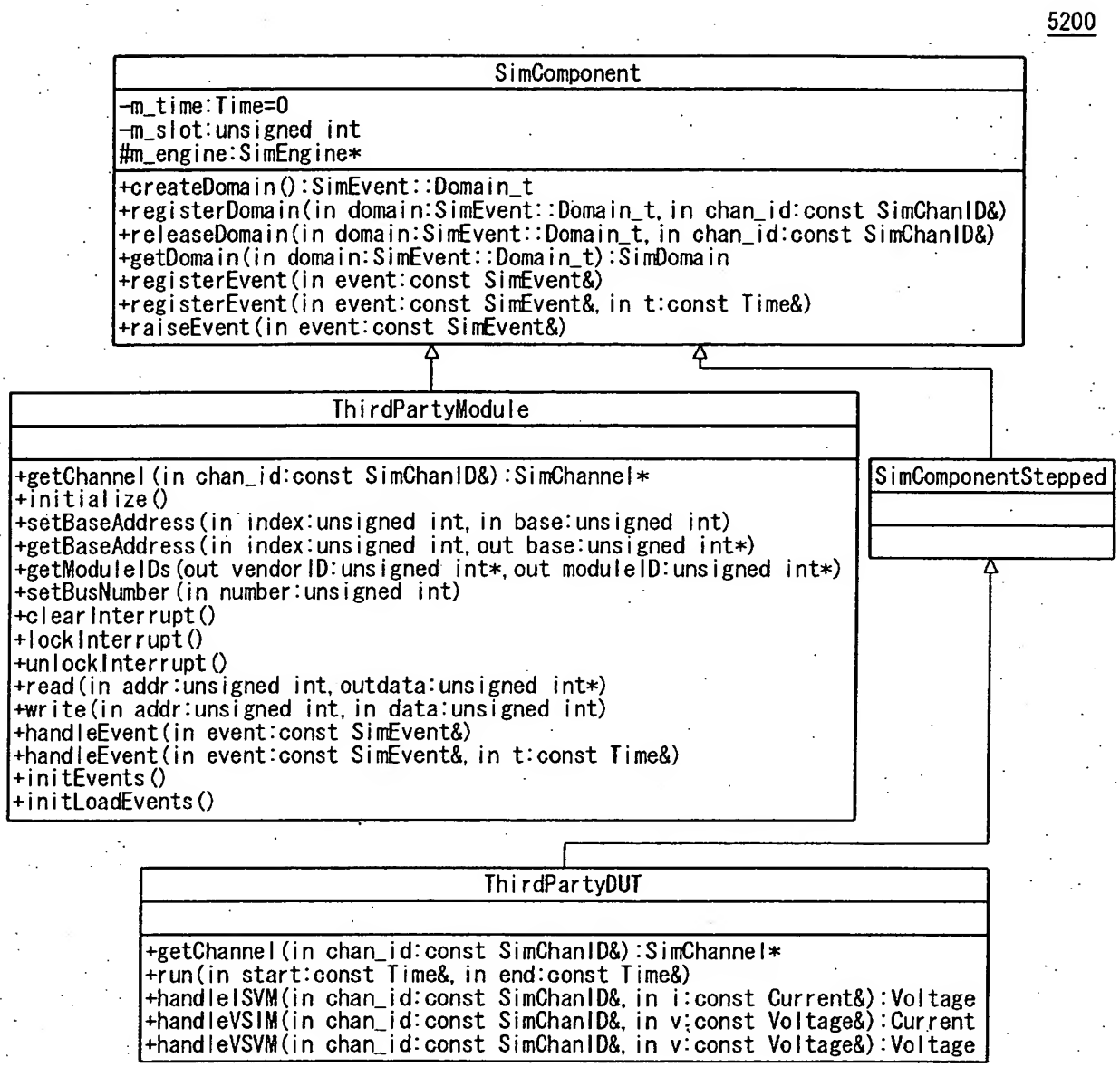
```
DUTModel "DUT1Sim"
{
  Waveform
  {
    Step 1-32;
  }
  DUT 1
  {
    Params
    {
      param1Name  "param1Value";
      param2Name  "param2Value";
    }
    PinConnections
    {
      L3.1    1    1.0 nS;
      L3.2    8    1.0 nS;
      L3.3    2    1.0 nS;
      L3.4    7    1.0 nS;
      L3.5    3    1.0 nS;
      L3.6    6    1.0 nS;
      L3.7    4    1.0 nS;
      L3.8    5    1.0 nS;
      L3.9    9    1.0 nS;
      L3.10  16    1.0 nS;
      L3.11  10    1.0 nS;
      L3.12  15    1.0 nS;
      L3.13  11    1.0 nS;
      L3.14  14    1.0 nS;
      L3.15  12    1.0 nS;
      L3.16  13    1.0 nS;
    }
  }
}
```

5120

~~図17~~ Fig. 17

```
DUTModel "DUT2Sim"
{
  Waveform
  {
    Slew 1-16 @ 2.0 V/nS;
  }
  DUT 2
  {
    Params
    {
      param1Name  "param1Value";
      param2Name  "param2Value";
    }
    PinConnections
    {
      L2.1   1;   # no delay specified means a delay of 0
      L2.2   2;
      L2.3   3;
      L2.4   4;
      L2.12  5;
      L2.13  6;
      L2.14  7;
      L2.15  8;
    }
  }
}
```

~~図18~~ Fig. 18


5200
SimComponent
-m_time:Time=0
-m_slot:unsigned int
#m_engine:SimEngine*
+createDomain():SimEvent::Domain_t
+registerDomain(in domain:SimEvent::Domain_t, in chan_id:const SimChanID&)
+releaseDomain(in domain:SimEvent::Domain_t, in chan_id:const SimChanID&)
+getDomain(in domain:SimEvent::Domain_t):SimDomain
+registerEvent(in event:const SimEvent&)
+registerEvent(in event:const SimEvent&, in t:const Time&)
+raiseEvent(in event:const SimEvent&)
ThirdPartyModule
+getChannel(in chan_id:const SimChanID&):SimChannel*
+initialize()
+setBaseAddress(in index:unsigned int, in base:unsigned int)
+getBaseAddress(in index:unsigned int, out base:unsigned int*)
+getModuleIDs(out vendorID:unsigned int*, out moduleID:unsigned int*)
+setBusNumber(in number:unsigned int)
+clearInterrupt()
+lockInterrupt()
+unlockInterrupt()
+read(in addr:unsigned int, outdata:unsigned int*)
+write(in addr:unsigned int, in data:unsigned int)
+handleEvent(in event:const SimEvent&)
+handleEvent(in event:const SimEvent&, in t:const Time&)
+initEvents()
+initLoadEvents()
SimComponentStepped
ThirdPartyDUT
+getChannel(in chan_id:const SimChanID&):SimChannel*
+run(in start:const Time&, in end:const Time&)
+handleISVM(in chan_id:const SimChanID&, in i:const Current&):Voltage
+handleVSIM(in chan_id:const SimChanID&, in v:const Voltage&):Current
+handleVSVM(in chan_id:const SimChanID&, in v:const Voltage&):Voltage

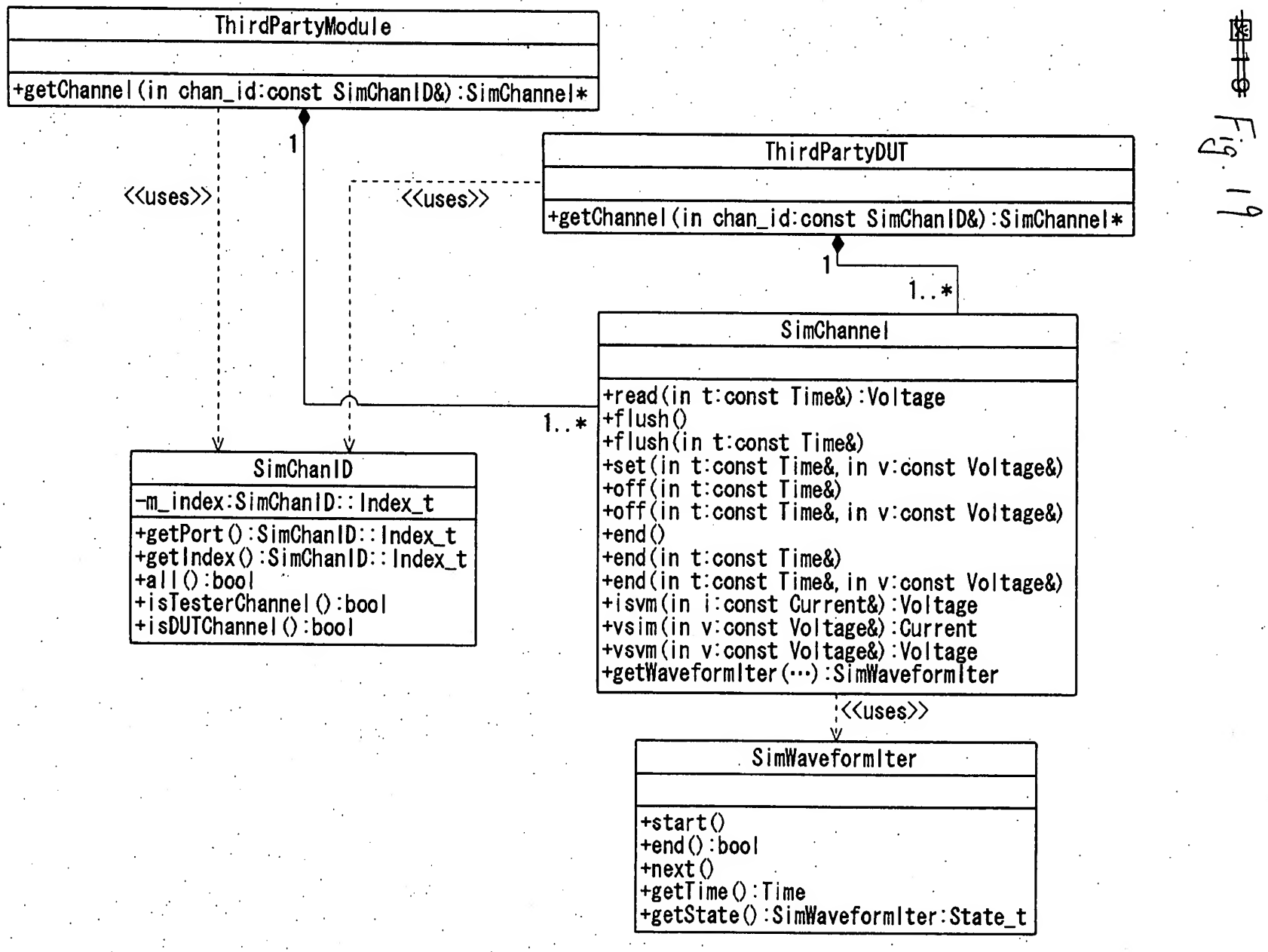

ThirdPartyModule
+getChannel(in chan_id:const SimChanID&):SimChannel*
1
<<uses>>
<<uses>>
ThirdPartyDUT
+getChannel(in chan_id:const SimChanID&):SimChannel*
1
1..*
SimChannel
+read(in t:const Time&):Voltage
+flush()
+flush(in t:const Time&)
+set(in t:const Time&, in v:const Voltage&)
+off(in t:const Time&)
+off(in t:const Time&, in v:const Voltage&)
+end()
+end(in t:const Time&)
+end(in t:const Time&, in v:const Voltage&)
+isvm(in i:const Current&):Voltage
+vsim(in v:const Voltage&):Current
+vsvm(in v:const Voltage&):Voltage
+getWaveformIter(…):SimWaveformIter
1..*
SimChanID
-m_index:SimChanID::Index_t
+getPort():SimChanID::Index_t
+getIndex():SimChanID::Index_t
+all():bool
+isTesterChannel():bool
+isDUTChannel():bool
<<uses>>
SimWaveformIter
+start()
+end():bool
+next()
+getTime():Time
+getState():SimWaveformIter:State_t

Fig. 19

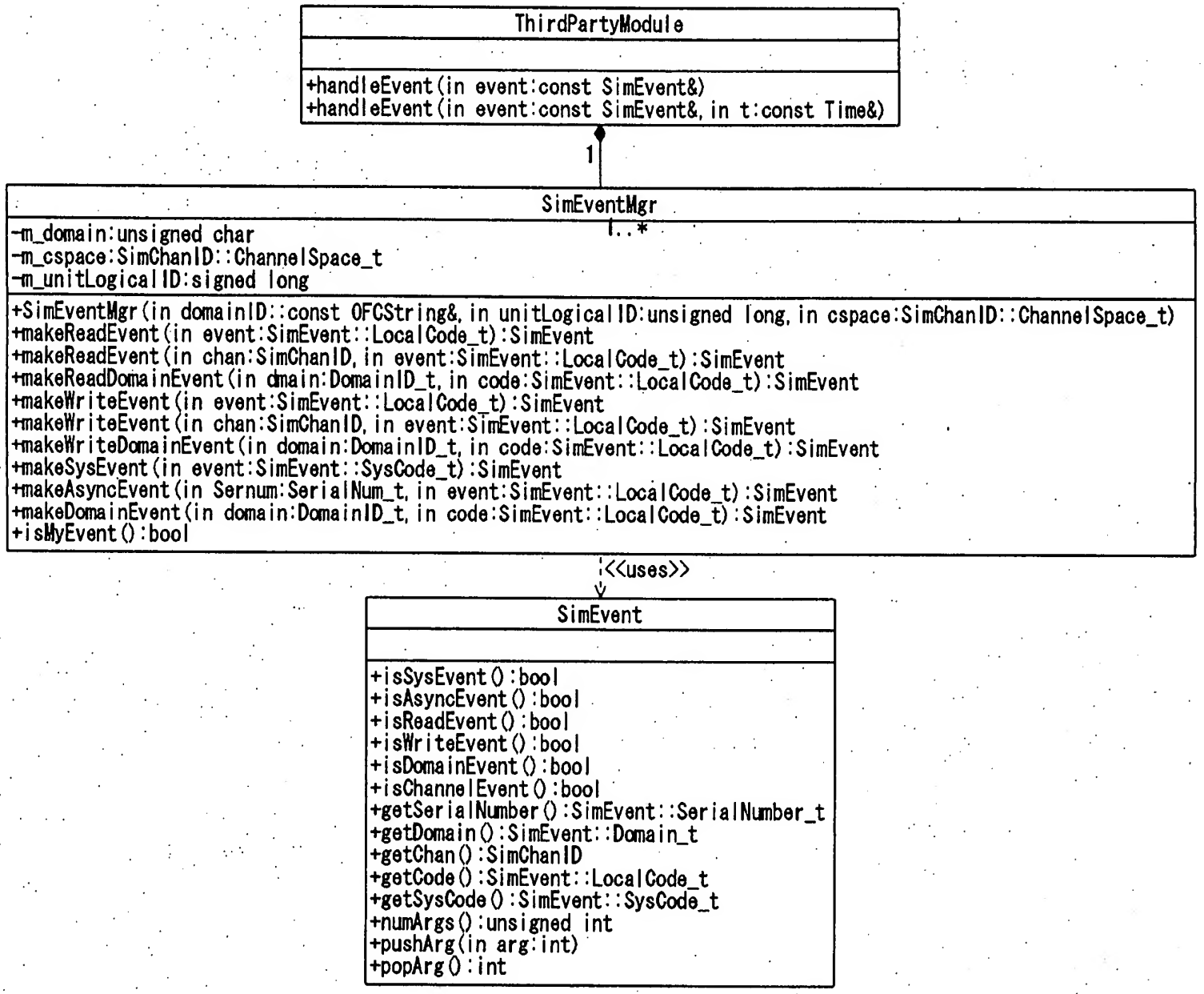

ThirdPartyModule
+handleEvent(in event:const SimEvent&)
+handleEvent(in event:const SimEvent&, in t:const Time&)
1
1..*
SimEventMgr
-m_domain:unsigned char
-m_cspace:SimChanID::ChannelSpace_t
-m_unitLogicalID:signed long
+SimEventMgr(in domainID::const OFCString&, in unitLogicalID:unsigned long, in cspace:SimChanID::ChannelSpace_t)
+makeReadEvent(in event:SimEvent::LocalCode_t):SimEvent
+makeReadEvent(in chan:SimChanID, in event:SimEvent::LocalCode_t):SimEvent
+makeReadDomainEvent(in domain:DomainID_t, in code:SimEvent::LocalCode_t):SimEvent
+makeWriteEvent(in event:SimEvent::LocalCode_t):SimEvent
+makeWriteEvent(in chan:SimChanID, in event:SimEvent::LocalCode_t):SimEvent
+makeWriteDomainEvent(in domain:DomainID_t, in code:SimEvent::LocalCode_t):SimEvent
+makeSysEvent(in event:SimEvent::SysCode_t):SimEvent
+makeAsyncEvent(in Sernum:SerialNum_t, in event:SimEvent::LocalCode_t):SimEvent
+makeDomainEvent(in domain:DomainID_t, in code:SimEvent::LocalCode_t):SimEvent
+isMyEvent():bool
<<uses>>
SimEvent
+isSysEvent():bool
+isAsyncEvent():bool
+isReadEvent():bool
+isWriteEvent():bool
+isDomainEvent():bool
+isChannelEvent():bool
+getSerialNumber():SimEvent::SerialNumber_t
+getDomain():SimEvent::Domain_t
+getChan():SimChanID
+getCode():SimEvent::LocalCode_t
+getSysCode():SimEvent::SysCode_t
+numArgs():unsigned int
+pushArg(in arg:int)
+popArg():int

Fig. 20

~~図21~~ Fig. 21

```
#include "OAI/sim/SimComponent.h"
#include "OAI/sim/SimChannel.h"
#include "OAI/sim/SimChanID.h"

#include "OAI/OFC/OFCString.h"

class MyBaseModule : public OASIS::SimComponent
{
public:

  // constructor
  MyBaseModule(unsigned int sernum, unsigned int prodrev,
               unsigned int port);

  // channel access
  OASIS::SimChannel* getChannel(const OASIS::SimChanID& chan_id);

  // do nothing for these initialization steps
  void initialize() { }
  void initLoadEvents() { }

  // ID
  unsigned int getSerialNumber() { return m_sernum; }
  unsigned int getProductRevision() { return m_prodrev; }

  // bus I/O
  void setBaseAddress(unsigned int index, unsigned int base);
  void getBaseAddress(unsigned int index, unsigned int* base);
  void getModuleIDs(unsigned int* vendorID, unsigned int* moduleID);
  void setBusNumber(unsigned int number);
  void clearInterrupt();
  void lockInterrupt();
  void unlockInterrupt();
  ClassCode_t getClassCode();
  void read(unsigned int addr, unsigned int* data);
  void write(unsigned int addr, unsigned int data);

  // pattern tracing features (unimplemented)
  void poll(unsigned int&, Time& t, unsigned int) { }
  void poll(unsigned __int64&, Time& t, unsigned int) { }
  void poll(unsigned double&, Time& t, unsigned int) { }
  void poll(unsigned OFCString&, Time& t, unsigned int) { }

protected:

  // internal types
  typedef unsigned int  port_t;
  typedef unsigned int  reg_t;
  typedef unsigned char vector_t;
  typedef unsigned char vecaddr_t

  // internal data
  OASIS::SimChannel       m_channels[8];
  reg_t                   m_sernum;
  reg_t                   m_prodrev;
  port_t                  m_port;
  reg_t                   m_base;
  reg_t                   m_period;
  reg_t                   m_edges[8];
  reg_t                   m_high;
  reg_t                   m_low;
  vector_t                m_pattern[100];
  vecaddr_t               m_patPtr;
  OASIS::SimEventMgr      m_evtMgr;
  bool                    m_intrLock;
};
```

~~図22~~ Fig. 22

```
#include "MyBaseModule.h"  // base class header

class MyDriveModule : public MyBaseModule
{
public:

  // constructor
  MyDriveModule(unsigned int sernum, unsigned int prodrev,
                unsigned int port);

  // access to vendor/module data
  void getModuleIDs(unsigned int* vendorID, unsigned int* moduleID);

  // initialize simulation
  void initEvents();

  // handle events
  void handleEvent(const OASIS::SimEvent& event);
  void handleEvent(const OASIS::SimEvent& event, const OASIS::Time& t);
};
```

~~図23~~ Fig. 23

```
void MyDriveModule::handleEvent(const OASIS::SimEvent& event,
                                const OASIS::Time& t)
{
  // t is the start of the cycle since that is the time we
  // registered for

  // write the edge into each channel
  for (chanaddr_t c = 0; c < 8; c++)
  {
    // write the edge to the output channel
    m_channels[c].set(t + m_edges[c] * 1.0e-12,
                      (m_pattern[m_patPtr] & 1 << c ? m_high : m_low)
                      * 1.0e-3);

    // terminate the write
    m_channels[c].end(t + m_period * 1.0e-12);
  }

  // register for the next start of cycle
  if (m_patPtr++ < 100)
  {
    registerEvent(m_evtMgr.makeWriteEvent(DRIVE_CYCLE),
                  t + m_period * 1.0e-12);
  }
}
```

~~図24~~ Fig. 24

```
#include "MyBaseModule.h"  // base class header

class MyStrobeModule : public MyBaseModule
{
public:

  // constructor
  MyStrobeModule(unsigned int sernum, unsigned int prodrev,
                 unsigned int port);

  // access to vendor/module data
  void getModuleIDs(unsigned int* vendorID, unsigned int* moduleID);

  // bus I/O
  void read(unsigned int addr, unsigned int* data);
  void write(unsigned int addr, unsigned int data);

  // initialize simulation
  void initEvents();

  // handle events
  void handleEvent(const OASIS::SimEvent& event);
  void handleEvent(const OASIS::SimEvent& event, const OASIS::Time& t);

private:

  // internal types
  typedef unsigned char chanaddr_t;

  // fail vector memory
  chanaddr_t m_fcm[10];
  vecaddr_t  m_fvm[10];
  reg_t      m_failCnt;
};
```

~~図25~~ Fig. 25

```
void MyStrobeModule::handleEvent(const OASIS::SimEvent& event,
                                 const OASIS::Time& t)
{
  // t is the end of the cycle since that is the time we
  // registered for

  // compute the start of cycle since all edges are relative to it
  OASIS::Time t0 = t - m_period * 1.0e-12;

  // strobe each channel
  for (chanaddr_t c = 0; c < 8; c++)
  {
    // make sure fail vector memory is not full
    if (m_failCnt == 10)
    {
      return;
    }

    // strobe the input channel
    Voltage v = m_channels[c].read(t0 + m_edges[c] * 1.0e-12);

    // test high
    if (m_pattern[m_patPtr] & 1 << c)
    {
      // fail
      if (v < m_high * 1.0e-3)
      {
        m_fcm[m_failCnt] = c;
        m_fvm[m_failCnt++] = m_patPtr;
      }
    }

    // test low
    else
    {
      // fail
      if (v > m_low * 1.0e-3)
      {
        m_fcm[m_failCnt] = c;
        m_fvm[m_failCnt++] = m_patPtr;
      }
    }

    // terminate the read
    m_channels[i].flush(t);
  }

  // register for the next end of cycle
  if (m_patPtr++ < 100)
  {
    registerEvent(m_evtMgr.makeReadEvent(STROBE_CYCLE),
                  t + m_period * 1.0e-12);
  }
}
```

~~図26~~ Fig. 26

```
#include "SimComponentStepped.h"
#include "SimChannel.h"
#include "SimChanID.h"
#include "OASISEngTypes.h" // for Time and Voltage

class MyDUTModel : public OASIS::SimComponentStepped
{
public:

  // constructor
  MyDUTModel(const OASIS::Voltage& vih, const OASIS::Voltage& vil,
             const OASIS::Voltage& voh, const OASIS::Voltage& vol);

  // get channels
  SimChannel* getChannel(const OASIS::SimChanID& chan_id);

  // run method
  void run(const OASIS::Time& t0, const OASIS::Time& tf);

private:

  // channels
  OASIS::SimChannel m_inputs[8];
  OASIS::SimChannel m_outputs[8];

  // levels
  OASIS::Voltage m_vih;
  OASIS::Voltage m_vil;
  OASIS::Voltage m_voh;
  OASIS::Voltage m_vol;
  OASIS::Voltage m_vz;
};
```

~~図27~~ Fig. 27

```
void MyDUTModel::run(const OASIS::Time& t0, const OASIS::Time& tf)
{
  // loop through each wire
  for (unsigned i = 0; i < 8; i++)
  {
    // loop through all the edges from the input channel
    for (OASIS::SimWaveformIter wf_iter
         = m_inputs[i].getWaveformIter(m_vih, m_vil, t0, tf);
         !wf_iter.end(); wf_iter.next())
    {
      // write edges with 1 nsec delay
      switch (wf_iter.getState())
      {
      case OASIS::SimWaveformIter::H:
        m_outputs[i].set(wf_iter.getTime() + 1.0e-9, m_voh);
      case OASIS::SimWaveformIter::L:
        m_outputs[i].set(wf_iter.getTime() + 1.0e-9, m_vol);
      case OASIS::SimWaveformIter::Z:
        m_outputs[i].set(wf_iter.getTime() + 1.0e-9, m_vz);
      }
    }

    // terminate write for each output channel
    m_outputs[i].end(tf);

    // flush each input channel
    m_inputs[i].flush(tf);
  }
}
```

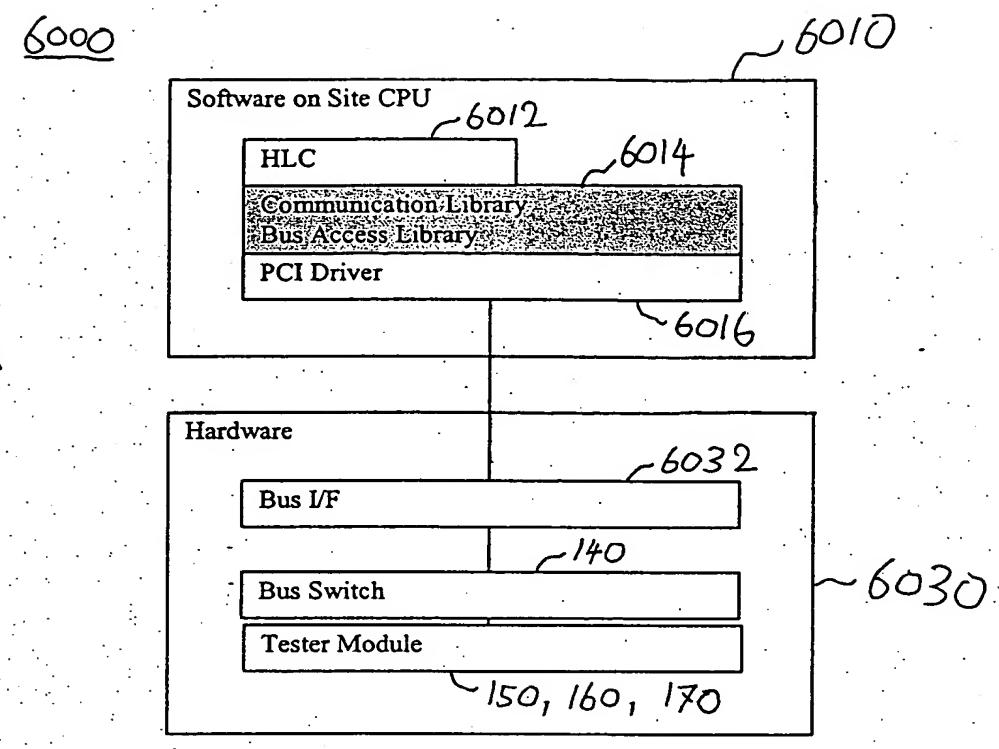


Fig. 28A

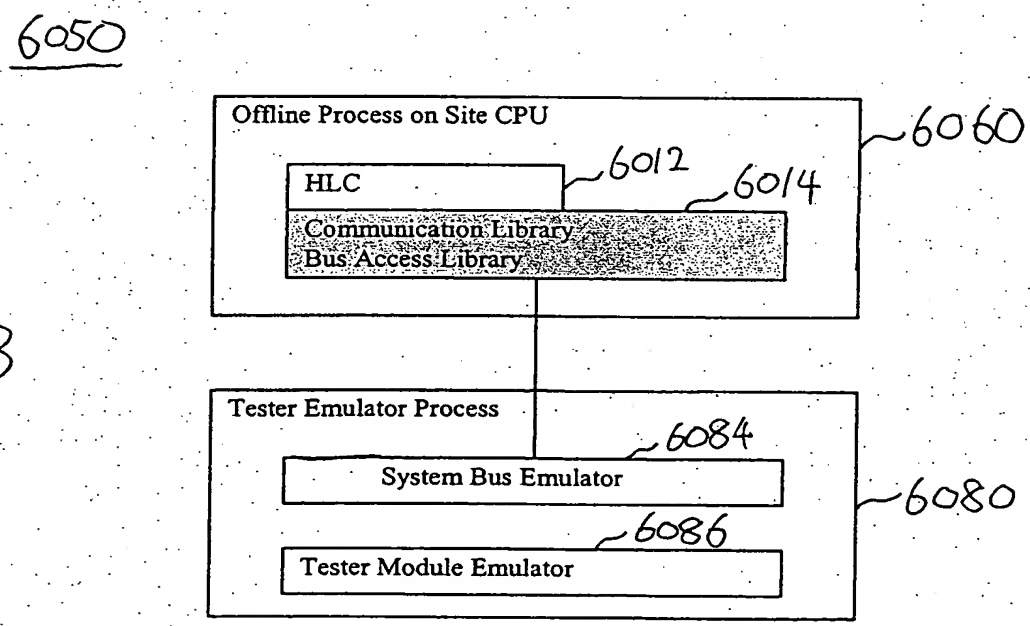


Fig. 28B